Handling The Client Request: Form Data

- One of the main motivations for building Web pages dynamically is so that the result can be based upon user input.

- This lecture will teach you how to access that input.

The Role of Form Data

- If you’ve ever used a search engine, visited an online bookstore, tracked stocks on the Web, or asked a Web-based site for quotes on plane tickets, you’ve probably seen funny-looking URLs like http://www.amazon.com/gp/search?rh=k%3Aj5ee%2Ci%3Astripbooks&keywords=j5ee&ie=UTF8&qid=1295541226
- The part after the question mark rh=k%3Aj5ee%2Ci%3Astripbooks&keywords=j5ee&ie=UTF8&qid=1295541226 is known as ***QUERY STRING*** *(form data or query data) and is the most common way to get information* from a Web page to a server-side program.
- Form data can be attached to the end of the URL after a question mark (as above) for GET requests; form data can also be sent to the server on a separate line for POST requests.

Form Basics

1. **Use the `FORM` element to create an HTML form.** Use the `ACTION` attribute to designate the address of the servlet or JSP page that will process the results; you can use an absolute or relative URL. For example:

   ```
   <FORM ACTION="...">...</FORM>
   ```

   If `ACTION` is omitted, the data is submitted to the URL of the current page.
2. **Use input elements to collect user data.** Place the elements between the start and end tags of the `FORM` element and give each input element a `NAME`. Textfields are the most common input element; they are created with the following.

   ```
   <INPUT TYPE="TEXT" NAME="...">
   ```
3. **Place a submit button near the bottom of the form.** For example:

   ```
   <INPUT TYPE="SUBMIT">
   ```

   When the button is pressed, the URL designated by the form's `ACTION` is invoked. With `GET` requests, a question mark and name/value pairs are attached to the end of the URL, where the names come from the `NAME` attributes in the HTML input elements and the values come from the end user. With `POST` requests, the same data is sent, but on a separate request line instead of attached to the URL.

Do I Use GET or POST?

- In HTML, one can specify two different submission methods for a form.
- The method is specified inside a FORM element, using the METHOD attribute.
- The difference between METHOD="GET" (the default) and METHOD="POST" is primarily defined in terms of form data encoding.
- There's a mixture of opinion on this one; some people say you should almost never use the GET method, due to its insecurity and limit on size; others maintain that you can use GET to retrieve information, while POST should be used whenever you modify data on the web server.

- One disadvantage of POST is that
  - pages loaded with POST cannot be properly book-marked,
  - whereas pages loaded with GET contain all the information needed to reproduce the request right in the URL.

- Extracting the needed information from this form data is traditionally one of the most tedious parts of server-side programming.
- First of all, before servlets you generally had to read the data one way for GET requests (in traditional CGI, this is usually through the QUERY_STRING environment variable) and a different way for POST requests (by reading the standard input in CGI).
- Second, you have to chop the pairs at the ampersands, then separate the parameter names (left of the equal signs) from the parameter values (right of the equal signs).
- Third, you have to *URL-decode the values: reverse the encoding that the browser* uses on certain characters. Alphanumeric characters are sent unchanged by the browser, but spaces are converted to plus signs and other characters are converted to %*XX*, *where XX is the ASCII (or ISO Latin-1) value of the character, in hex. For* example, if someone enters a value of "~hall, ~gates, and ~mcnealy" into a textfield with the name users in an HTML form, the data is sent as

  "users=%7Ehall%2C+%7Egates%2C+and+%7Emcnealy",

  and the server-side program has to reconstitute the original string.
- Finally, the fourth reason that it is tedious to parse form data with traditional server-side technologies is that values can be omitted

  (e.g., "param1=val1&param2=&param3=val3")

  or a parameter can appear more than once

  (e.g., "param1=val1&param2=val2&param1=val3"),

  so your parsing code needs special cases for these situations.

- **Fortunately, servlets help us with much of this tedious parsing.**

Reading Form Data from Servlets

- One of the nice features of servlets is that all of this form parsing is handled automatically.
- You call request.getParameter to get the value of a form parameter.
- You can also call request.getParameterValues if the parameter appears more than once, or
- you can call request.getParameterNames if you want a complete list of all parameters in the current request.
- In the rare cases in which you need to read the raw request data and parse it yourself, call getReader or getInputStream.

Java EE Documentation

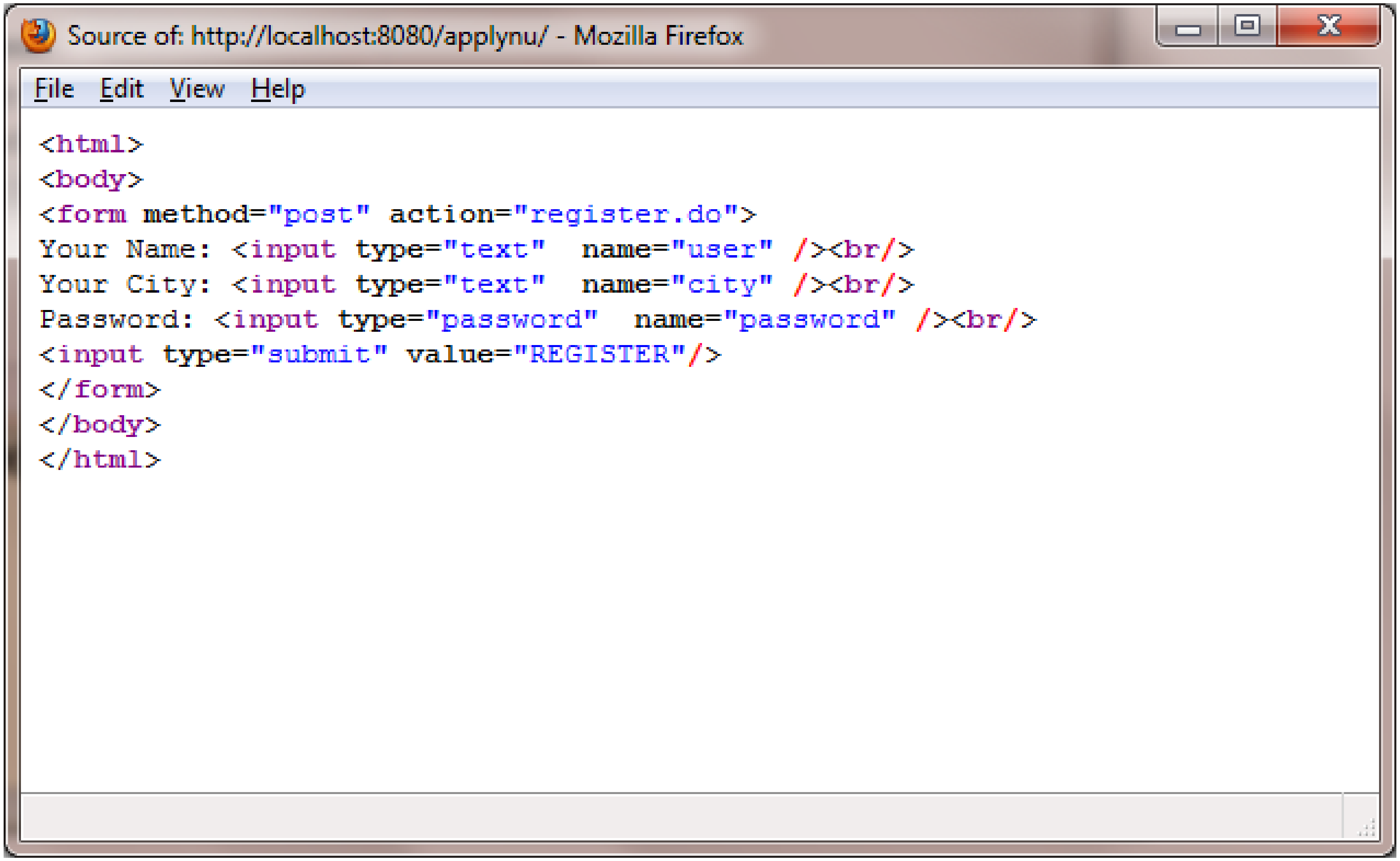
java.lang.String	getParameter(**java.lang.String name)** Returns the value of a request parameter as a String, or null if the parameter does not exist.
java.util.Map<java.lang.String,java.lang.String[ ]>	getParameterMap( ) Returns a java.util.Map of the parameters of this request.
java.util.Enumeration<java.lang.String>	getParameterNames( ) Returns an Enumeration of String objects containing the names of the parameters contained in this request.
java.lang.String[ ]	getParameterValues(**java.lang.String name)** Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist.

Reading Single Values: getParameter

- To read a request (form) parameter, you simply call the getParameter method of HttpServletRequest, supplying the case-sensitive parameter name as an argument.
- You supply the parameter name exactly as it appeared in the HTML source code, and you get the result exactly as the end user entered it; any necessary URL-decoding is done automatically.
- Unlike the case with many alternatives to servlet technology, you use getParameter exactly the same way when the data is sent by GET as you do when it is sent by POST; the servlet knows which request method the client used and automatically uses the appropriate method to read the data.
- An empty String is returned if the parameter exists but has no value (i.e., the user left the corresponding textfield empty when submitting the form), and null is returned if there was no such parameter.
- Parameter names are case sensitive, so the followings are *not* interchangeable.

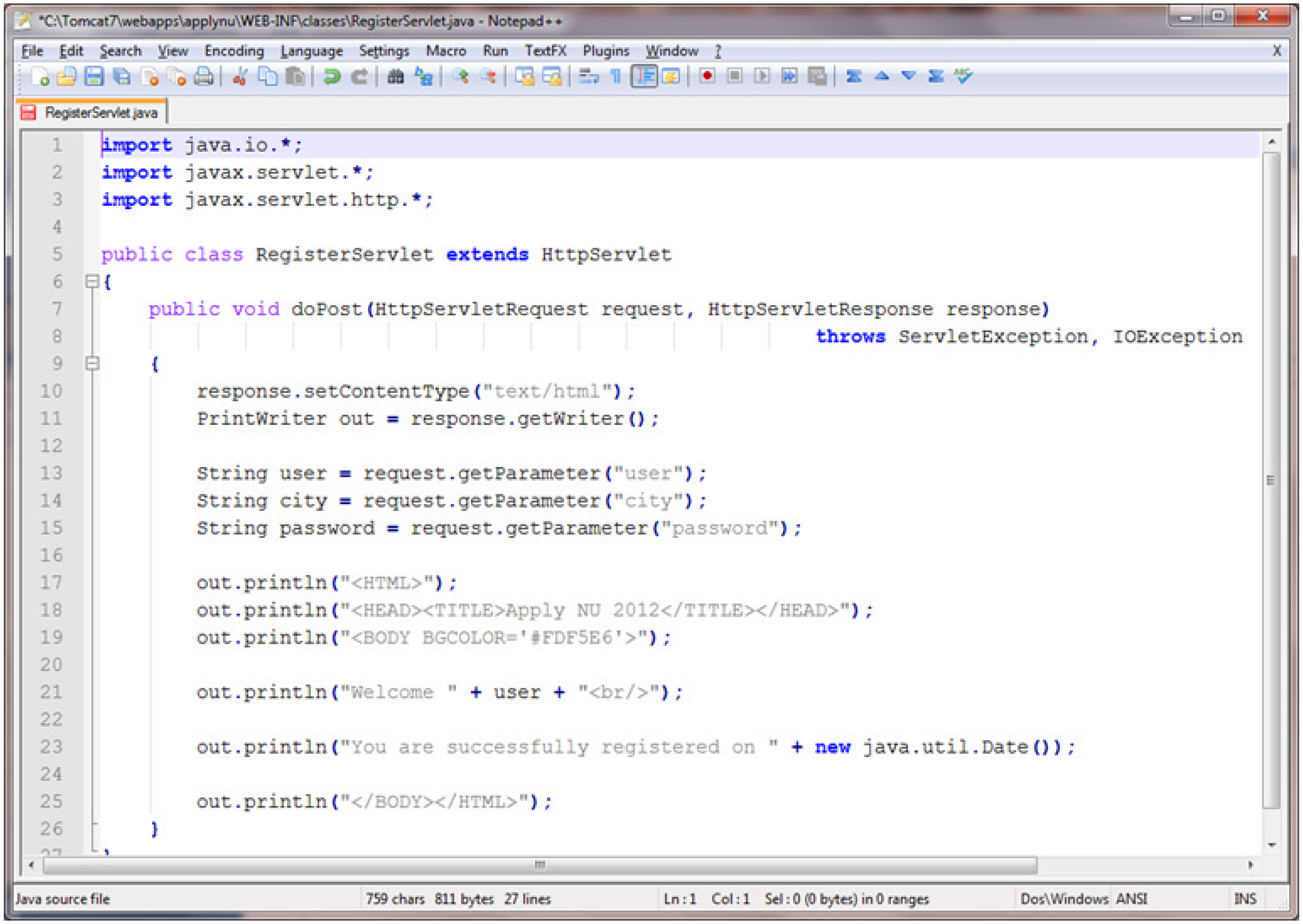
  request.get*Parameter("Param1") and request.getParameter("param1")*

In-Class Exercise



```
<html>
<body>
<form method="post" action="register.do">
Your Name: <input type="text"  name="user" /><br/>
Your City: <input type="text"  name="city" /><br/>
Password: <input type="password"  name="password" /><br/>
<input type="submit" value="REGISTER"/>
</form>
</body>
</html>
```

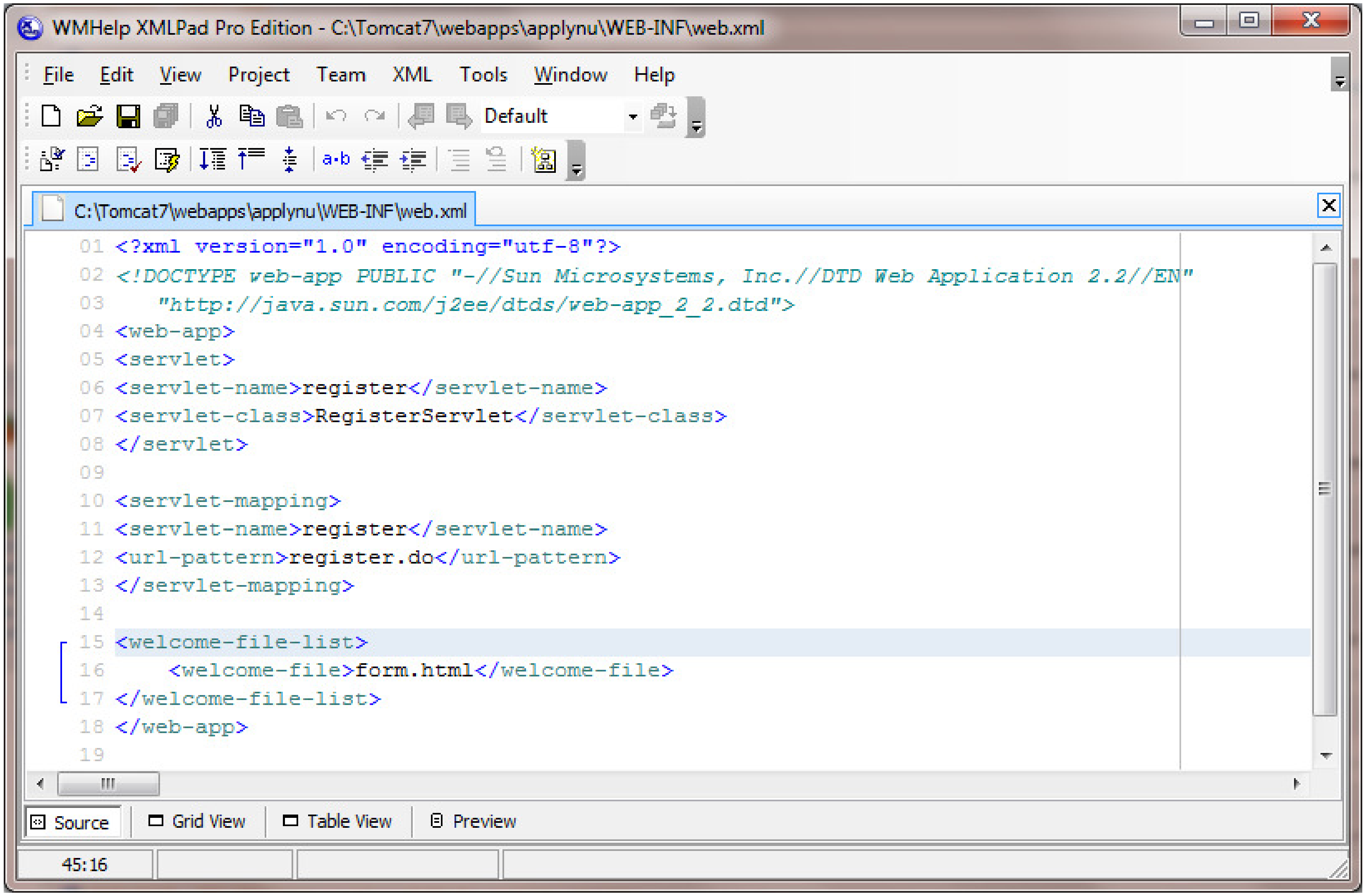
Servlet to handle the request


*C:\Tomcat7\webapps\applynu\WEB-INF\classes\RegisterServlet.java - Notepad++
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
RegisterServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class RegisterServlet extends HttpServlet
{
    public void doPost(HttpServletRequest request, HttpServletResponse response)
                                                throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String user = request.getParameter("user");
        String city = request.getParameter("city");
        String password = request.getParameter("password");

        out.println("<HTML>");
        out.println("<HEAD><TITLE>Apply NU 2012</TITLE></HEAD>");
        out.println("<BODY BGCOLOR='#FDF5E6'>");

        out.println("Welcome " + user + "<br/>");

        out.println("You are successfully registered on " + new java.util.Date());

        out.println("</BODY></HTML>");
    }
```

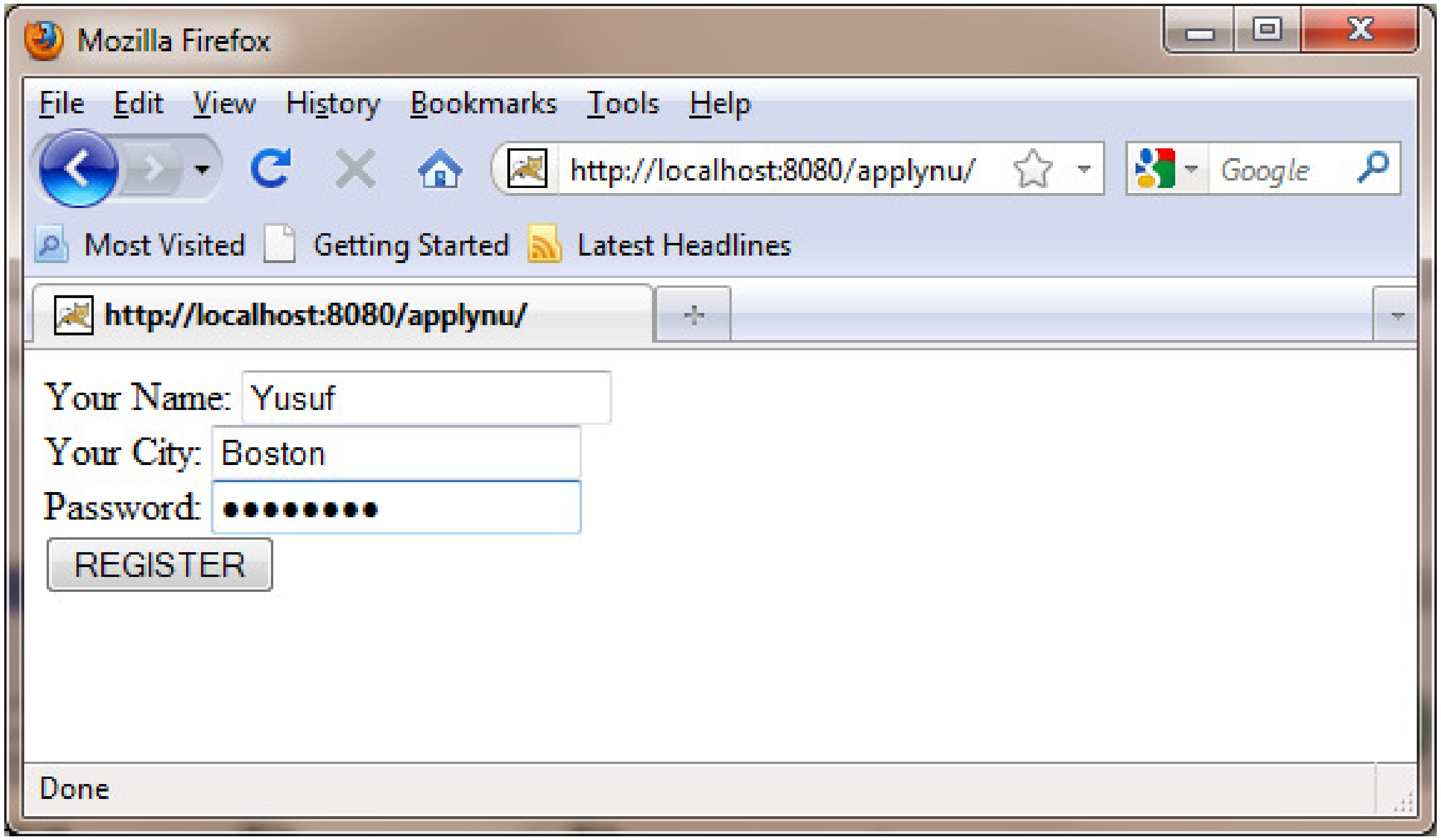
Java source file
759 chars 811 bytes 27 lines
Ln : 1 Col : 1 Sel : 0 (0 bytes) in 0 ranges
Dos\Windows ANSI
INS

Deployment descriptor – web.xml


WMHelp XMLPad Pro Edition - C:\Tomcat7\webapps\applynu\WEB-INF\web.xml
File Edit View Project Team XML Tools Window Help
Default
C:\Tomcat7\webapps\applynu\WEB-INF\web.xml
Source Grid View Table View Preview
45:16

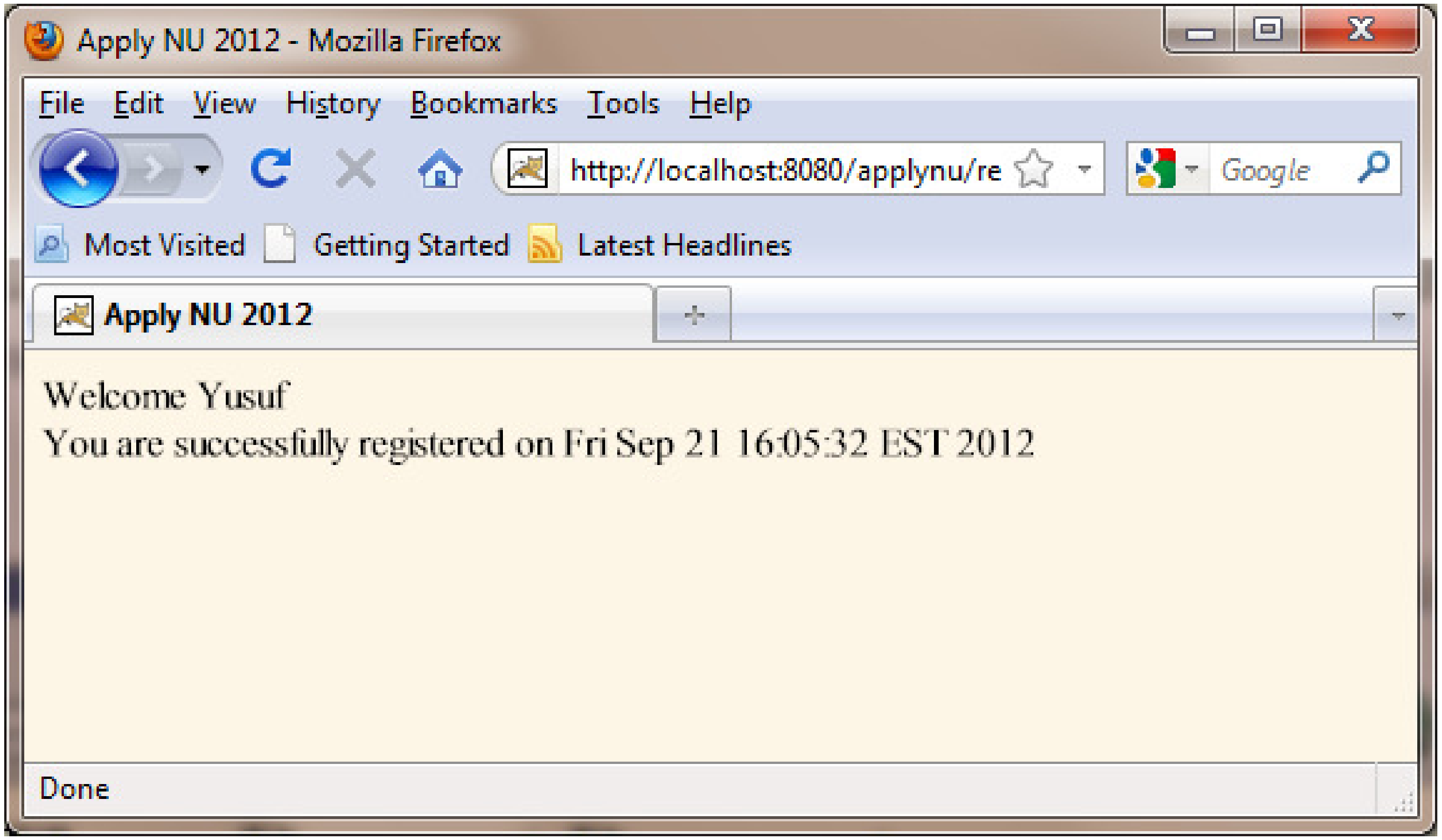
```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
   "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
<servlet>
<servlet-name>register</servlet-name>
<servlet-class>RegisterServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>register</servlet-name>
<url-pattern>register.do</url-pattern>
</servlet-mapping>

<welcome-file-list>
    <welcome-file>form.html</welcome-file>
</welcome-file-list>
</web-app>

```

Running the servlet


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/applynu/
Google
Most Visited Getting Started Latest Headlines
http://localhost:8080/applynu/

Your Name: Yusuf
Your City: Boston
Password: •••••••••

Generating HTML dynamically


Apply NU 2012 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/applynu/re
Google
Most Visited Getting Started Latest Headlines
Apply NU 2012
Done

Welcome Yusuf
You are successfully registered on Fri Sep 21 16:05:32 EST 2012

Reading Multiple Values: getParameterValues

- If the same parameter name might appear in the form data more than once, you should call getParameterValues (which returns an array of strings) instead of getParameter (which returns a single string).
- The return value of getParameterValues is null for nonexistent parameter names and is a one-element array when the parameter has only a single value.
- Checkboxes, and multiselectable list boxes (i.e., HTML SELECT elements with the MULTIPLE attribute set repeat the parameter name for each selected element in the list return multiple values.

In-Class Exercise

- Reading Multiple Value

Looking Up Parameter Names: getParameterNames and getParameterMap

- Most servlets look for a specific set of parameter names; in most cases, if the servlet does not know the name of the parameter, it does not know what to do with it either.
- So, your primary tool should be getParameter.
- However, it is sometimes useful to get a full list of parameter names. The primary utility of the full list is debugging, but you occasionally use the list for applications where the parameter names are very dynamic.
- For example, the names themselves might tell the system what to do with the parameters (e.g., row-1-col-3-value), the system might build a database update assuming that the parameter names are database column names, or the servlet might look for a few specific names and then pass the rest of the names to another application.
- Use getParameterNames to get this list in the form of an Enumeration, each entry of which can be cast to a String and used in a getParameter or getParameterValues call. If there are no parameters in the current request, getParameterNames returns an empty Enumeration (not null).
- An alternative to getParameterNames is getParameterMap. This method returns a Map: the parameter names (strings) are the table keys and the parameter values (string arrays as returned by getParameterNames) are the table values.

In-Class Exercise

- Reading Parameter Names